Success Factors for Adoption of Real-Time Java

Kelvin Nilsen, CTO

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **APR 2010**	2. REPORT TYPE	3. DATES COVERED **00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE **Success Factors for Adoption of Real-Time Java**	5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Atego,5930 Cornerstone Court West, Suite 250,San Diego,CA,92121**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Presented at the 22nd Systems and Software Technology Conference (SSTC), 26-29 April 2010, Salt Lake City, UT. Sponsored in part by the USAF. U.S. Government or Federal Rights License

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**32**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Why this topic?

- A small number of companies have reported great success with the adoption of embedded/real-time Java

  Two-fold productivity improvement during development of new functionality

  Five- to ten-fold productivity improvement during software maintenance and reuse activities

  Improved functionality and fewer bugs

- But some highly visible attempts to adopt embedded/real-time Java have quietly fallen off the radar screen
- And overall adoption of Java in the embedded/real-time marketplace has been relatively slow
- We would like to understand why some projects fail and others succeed, in order to improve the likelihood of success, in hopes of enabling faster market expansion

Sample Successes with a Soft Real-Time Java Profile

- Use
  Typi
- The com
- The ope
- prov
  Hun
  Millio
- Cur

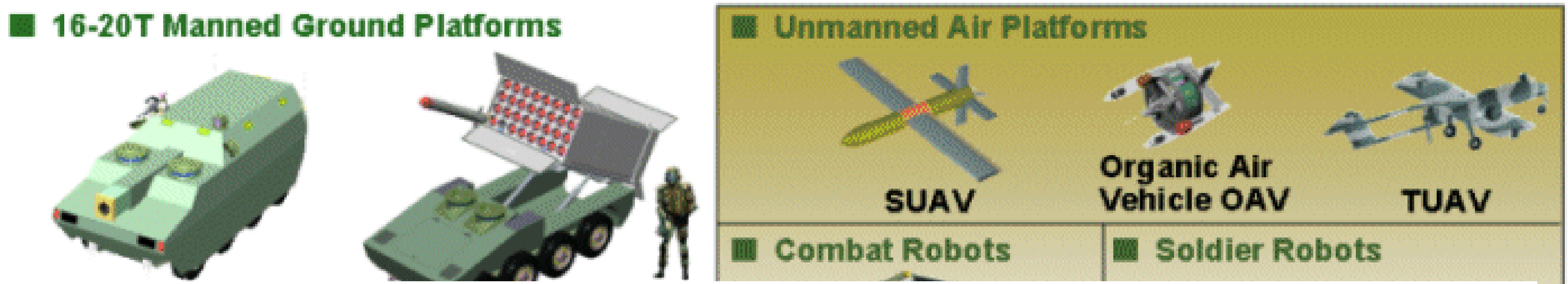
Software modernization project implements improved user interfaces, new communication protocols, and new functionality in Java

Projects that have fallen off the radar

- Project “Golden Gate”: a collaboration between Sun Labs, Carnegie Mellon-West, and NASA Jet Propulsion Labs to develop a real-time Java software architecture called MDS (Mission Data System).

- An integration of the Real-Time Specification of Java within the “HotSpot Java performance engine” for a customer who is “going to use this implementation for an application system that they’re developing that controls fossil fuel power plants.”

Two Highly Visible Projects Recently Downsized



- Raytheon doing DDG-1000 (a.k.a. Zumwalt) software using real-time Java technologies provided by IBM
- Boeing doing FCS software using PERC Ultra VM technologies
- Some signs of potential trouble on both of these

DDG-1000 “canceled” after only 2 ships funded

FCS: we are concerned by lack of direct communication with FCS engineers

Both projects have faced congressional/Pentagon budget cuts

Most common factors leading to lack of success

- Unclear, unrealistic or misguided objectives, often leading to …
  - Inadequate or misdirected risk mitigation and project planning
- Common misunderstandings regarding technology strengths and weaknesses

Real-time Java runs faster than normal Java

The risk/benefits analysis widely accepted for traditional Java is equally applicable to “real-time Java”

Based on experience with traditional Java, expect real-time Java developers to be 2x as productive and software maintainers to be 5-10x as productive

The key risk mitigation topic: determine the lower limits of scheduling latency that can be achieved with real-time Java

You can “easily” teach a traditional Java developer to become a real-time Java developer by teaching him or her the “real-time Java API”

All real-time Java offerings are “the same”

By adhering to the RTSJ “standard”, we assure that real-time code is portable, interoperable, composable, and maintainable

Proactive steps to improve likelihood of success

- Address potential issues with internal “politics”
- Establish clear, measurable, and realistic objectives
- Select technologies that suit the needs of established objectives
- Obtain appropriate training
- Give proper attention to architecture and design considerations
- Plan for maintenance, evolution, and reuse of software
- Establish clear separation of concerns
- Design efficient and robust interfaces between legacy and Java software

Internal Politics

- Issues:

  Especially in larger more established organizations, powerful established players may fight to “protect turf”.

  There are good reasons for everyone to exercise caution with “radical” new software approaches.

  However, watch out for the threat of torpedo attacks on your efforts.

- Example:

  One large aerospace customer chose to build a hybrid system including the integration of Java and C code.

  The C team represented established strength and political influence within the organization. The Java team represented new, younger talent.

  As the project evolved, the Java team sensed lack of cooperation, finger pointing, “frequent” blame (unfairly) directed towards the choice to use Java.

  Suspicion: established C developers felt threatened by prospect of Java success.

Establish clear, measurable, realistic objectives

- "Good" reasons to pursue Java might include:
  - Improve portability of software
  - Reduce costs of development
  - Accelerate availability of new capabilities
  - Exploit off-the-shelf and open-source software capabilities
  - Reduce errors in developed software
  - Increase generality and flexibility of software
  - Enable increased future reuse of software
  - Reduce long-term software maintenance costs
  - Improve security of software systems
- Note: the choice to use Java or real-time Java does not by itself guarantee any of the above
- You must use Java in an effective way in order to achieve objectives
- If you don't plan to satisfy objectives, and don't measure results against your plan in order to "correct your course" along the way, you're likely to fail

Select technologies that suit your objectives

- Java comes in many flavors
- Java built-in libraries, and 3rd party Java software components represent many different tradeoffs
- Selecting the appropriate Java development tools also represents an important leverage point
- Do you need real-time? Do you need embedded? Do you need a "Java" logo? What tradeoffs will you accept?

  Speed vs. footprint

  Predictability and reliability vs. optimal throughput

  Ease of development vs. cost of deployment

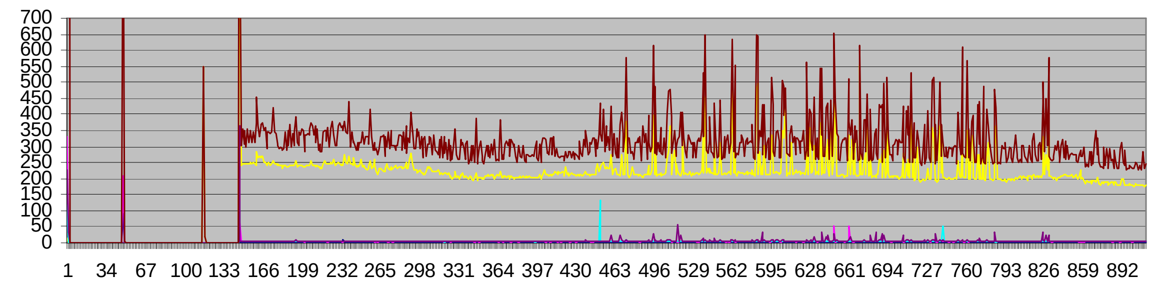
  Cost of development vs. cost of maintenance

Real-Time Horses for Courses

	HotSpot Java	C	PERC Ultra	PERC Pico
Min (ns)	173	280	519	314
Max (ns)	8,089	571	1,060	633
Mean (ns)	294	360	639	392
Std deviation (ns)	228.7896	25.52465	48.66415	29.8434
Total Virtual Memory (MB)	253	2.32	24	2.5

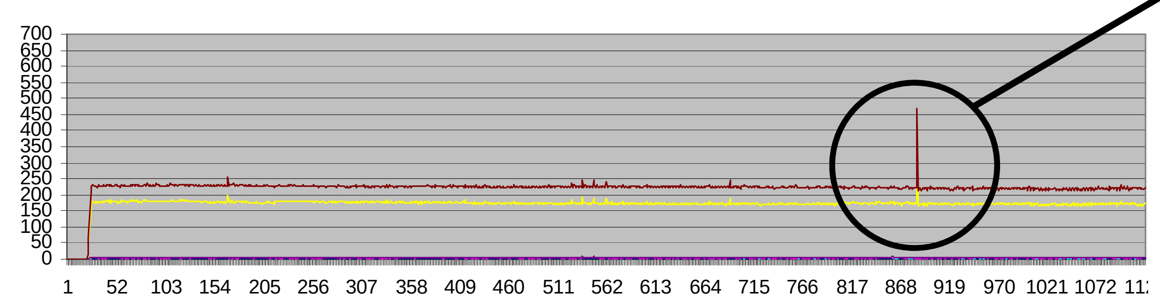
Obtain appropriate training

Results of moving traditional Java to real-time VM



Improvements after one week of training/consulting

operating system interference



Setting proper priorities is much more important with a real-time VM: “One week of on-site consulting saved six months of development time!”

Traditional Java virtual machines

- The Java platform was designed to be portable and scalable
- It was not designed to extend the benefits of portability and scalability into the realm of real-time software
  - Only ten thread priorities, and priorities are only a suggestion
  - Synchronization does not implement priority inversion avoidance
  - The order of threads on an Object.wait() queue is unspecified
  - The order of threads on the ready list is unspecified
  - Garbage collection happens at unpredictable times, consuming unpredictable amounts of CPU time, and does not guarantee to find all garbage, nor to defragment the available free pool
  - Common operations may have surprising effects (e.g. entering a mutual exclusion region might force allocation of memory, which might trigger garbage collection)
- The Real-Time Specification for Java (RTSJ), published in 2001, improves upon the traditional JVM specification, but still does not guarantee portability or scalability
  - Note that RTSJ is not part of the “standard edition” Java

Thread Priorities

- Select an appropriate real-time virtual machine to enable enhancements to the traditional Java thread priority model. For example:

  More than ten priorities can be supported. In case legacy software assumes only ten priorities, multiple real-time thread priorities can be mapped to a single “Java” priority (e.g. real-time priorities 11-28 all masquerade as legacy Java priority 10)

  Strict priority dispatching is offered, rather than treating priority as only a “suggestion”

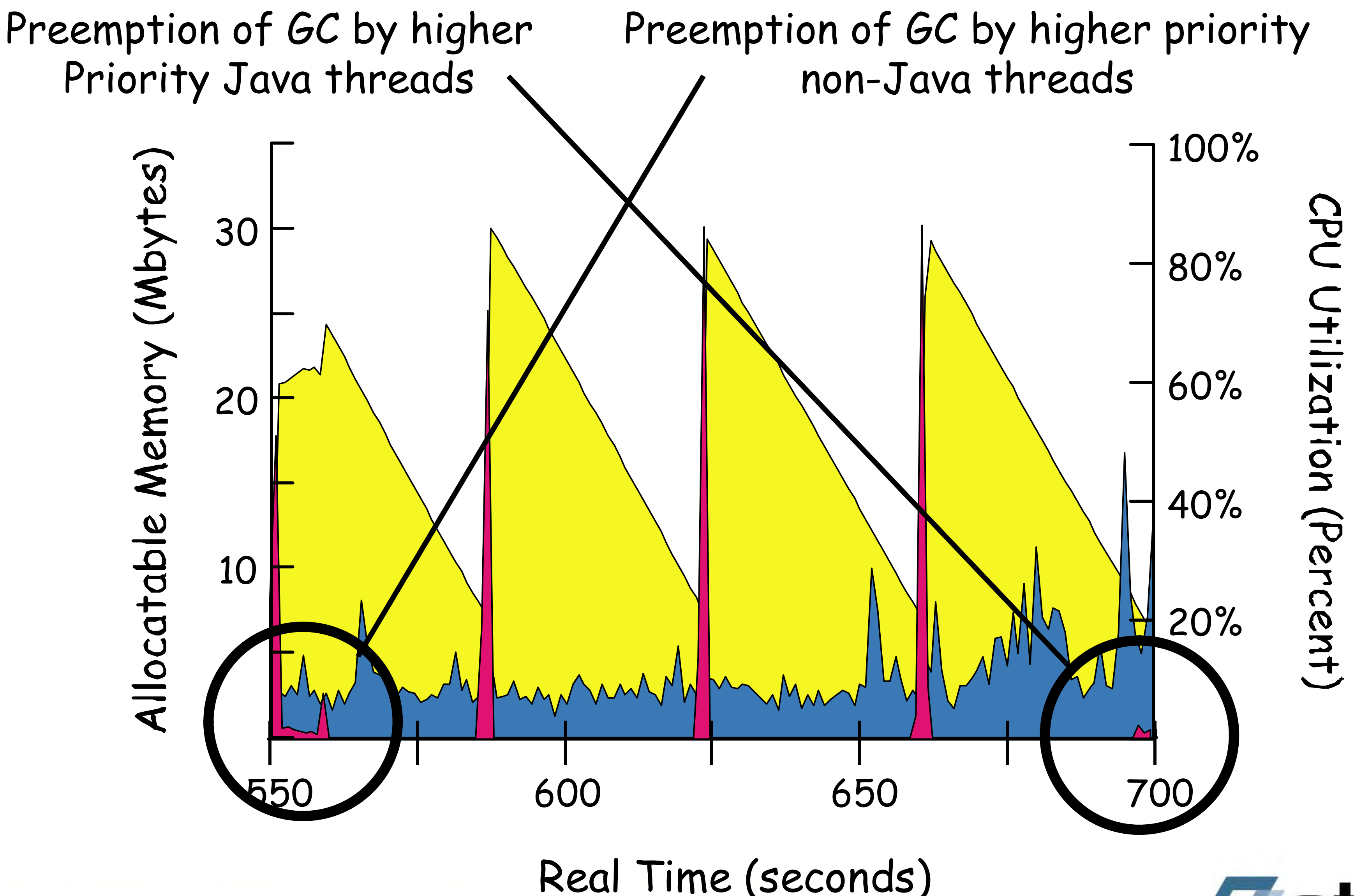
  Java priorities can be mapped to underlying operating system priorities to enable rate monotonic analysis of mixed language real-time systems

  CPU time accounting can be reported for each thread, and for each priority level

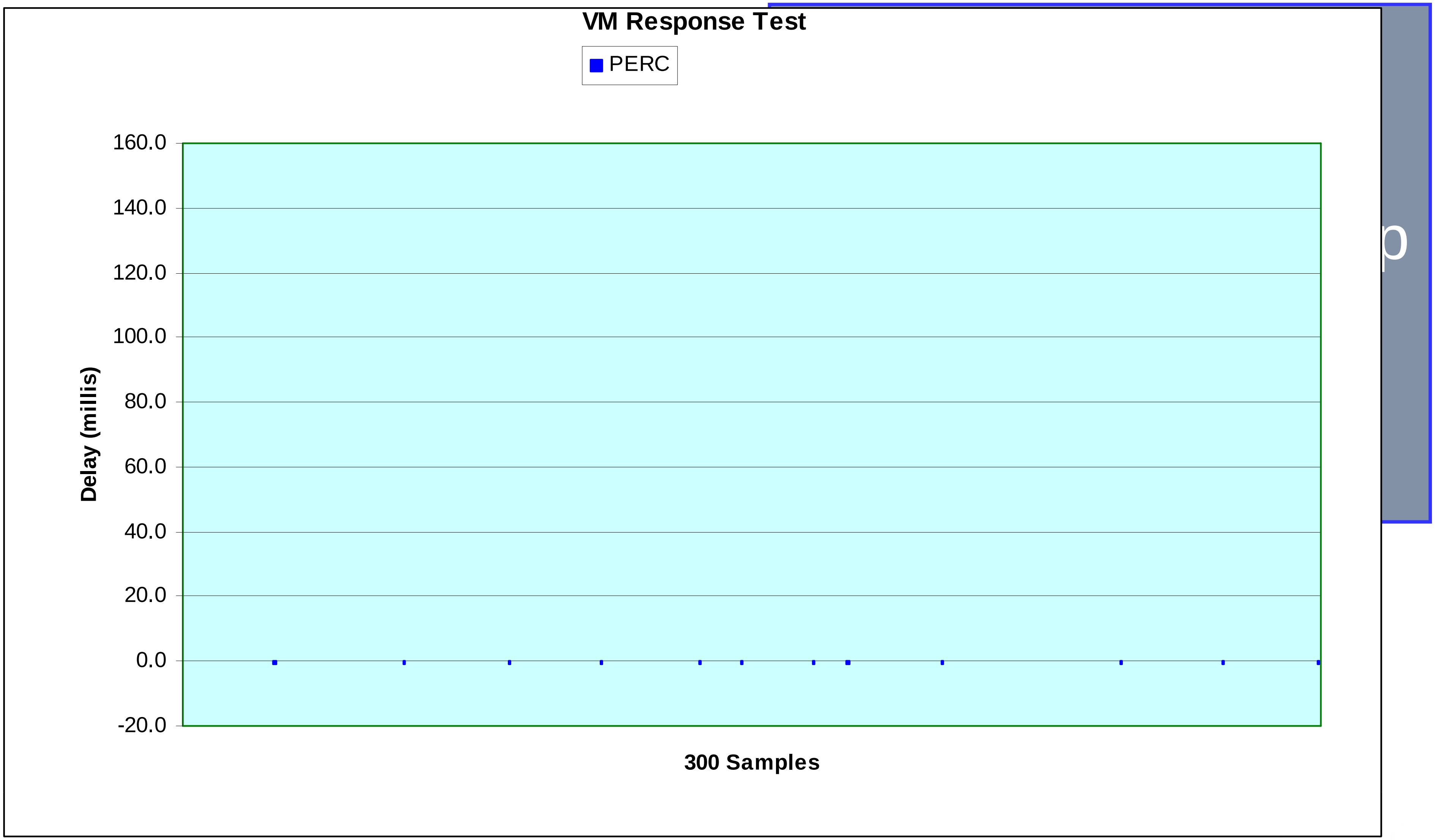
The Java Run-time

- Appropriate real-time virtual machines offer enhanced synchronization capabilities:
  - Maintain all thread queues in priority FIFO order
  - Implement priority inheritance for all Java synchronization
  - Detect circular lock dependencies, which indicate deadlock
  - Priority ceiling emulation for static hard real-time code
  - On-line monitoring capabilities to identify which threads hold which locks, and which threads are blocked awaiting access to particular locks
- Real-time garbage collection can be preemptible, incremental, accurate, defragmenting, paced

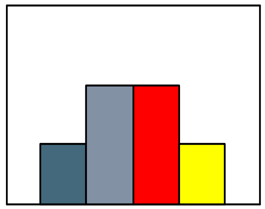
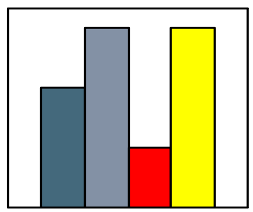
Preemption of Garbage Collection



Classic VM running VM Response Test

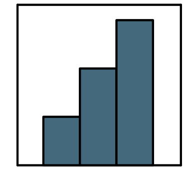


Static Linking and Ahead-of-Time Translation

	Dynamic Loading	Static Linking
Interpreted		Not "traditional Java"
JIT Compiled		Not applicable
AOT Compiled	Not "traditional Java"	Not "traditional Java"

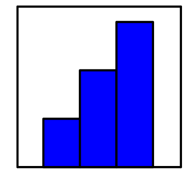
ROM Size

Large
Med
Small



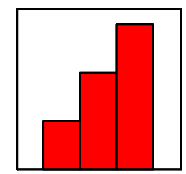
RAM Size

Large
Med
Small



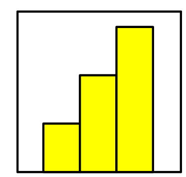
Start Speed

Fast
Med
Slow



Exec Speed

Fast
Med
Slow



Library Selection

- Though the power of standard-edition Java libraries is an important part of Java's appeal, deploying with full libraries is often undesirable:

  Memory/cost constraints may demand library subsetting

  Limitations of the embedded device may make certain libraries irrelevant (e.g. some devices have no file system, network connections, or GUI)

  Stability considerations may demand that certain systems freeze with older library versions

  Security and/or safety certification concerns may require exclusion of certain libraries

- Most vendors of embedded system virtual machines offer greater flexibility for library subsetting than the traditional Java license
- Some vendors provide tools to assist in analyzing interdependencies between libraries and "robustly stub out" omitted functionality

Eliminating Garbage Collection

- In systems that have severe memory or performance constraints, or that must be certified to the highest levels of safety rigor, garbage collection may be eliminated entirely

As an object-oriented programming language, it is difficult (and unnatural) to write programs that do not allocate any memory at all

In the absence of tracing garbage collection, temporary objects can be safely allocated on the run-time stack

  - The RTSJ introduces an explicit mechanism known as “scopes”
  - A simpler and safer specialization of RTSJ scopes is being developed in JSR-302 (safety-critical Java specification)
  - Certain vendors provide tools to prove that scope allocations do not introduce dangling pointers, and to automate the calculation of scope sizes

Direct Access to Hardware Devices

- As a high-level programming language, Java strives to protect programmers from low-level details, like device access

Real-time virtual machines provide various mechanisms to allow access to device interfaces, including:

  - java.nio.MappedByteBuffer maps a region of physical memory to a Java data structure
  - Some real-time virtual machines have specialized APIs that directly access physical memory locations, the implementation of which is “in-lined” by the JIT compiler

Interrupt Handlers and Device Drivers in Java

- Most modern device drivers are interrupt driven
- Interrupt handlers are dispatched by hardware rather than software
- Traditional Java is not capable of running first-level interrupt handlers
- Code written in a style of “hard real-time Java”, avoiding all dependencies on garbage collection, can be dispatched by the interrupt handling hardware

Architecture and Design Considerations

- Note that Java is most relevant to larger, more complex systems that are beyond the “comprehension” of a single developer
- Abstraction allows teams of developers to divide and conquer
- But everyone must “see” the same abstractions
- Don’t start development until you have a general architecture and design
- Study available libraries, frameworks, and 3rd party offerings in order to determine relevance
- Establish rules for interaction and separation of concerns between independently developed components

Plan for maintenance, evolution, reuse

- The system architects and designers have greatest responsibility for organizing software so that it can evolve
- Object-oriented notations allow designers to specify component interfaces in ways that facilitate software reuse and evolution
- Establish practices that encourage communication, enable feedback from developers to designers, and allow refinements from designers to developers
- Spiral development and extreme programming offer many benefits

Establish Clear Separation of Concerns

- Expose internal details on a need-to-know basis
- Give proper attention to use of package, private, final restrictions during design
- Many real-time considerations violate traditional objectives of abstraction, for example:

  How much memory is required to incorporate this component?

  How much CPU time is required to invoke this service?

  What demands does this component impose on the real-time garbage collector?

- Suggestions:

  Architects and designers need to address real-time considerations as they specify the APIs, documentation, and other artifacts associated with specific real-time components

  To aid maintenance and composition of components, consider wrapping components within a framework that assesses and represents resource needs

Real-time attributes of software components

- For active components, a real-time Java component framework might require individual components to identify themselves with regards to:

  How many threads? What are their response time constraints? What is their demand for CPU time?

  How much live memory is retained by this application?

  What is the pace at which garbage memory is created by this application?

- For passive components, the component might report:

  How much CPU time and memory is required to invoke each service?

- Resource requirements can be determined analytically or approximated empirically – to facilitate maintenance and reuse, bundle resource determination with the code

- If resource needs for certain code are not known, isolate this code (at lower priority or on distinct virtual machines)

  Note: a low-priority thread that allocates memory can hinder progress of high-priority threads that need to allocate memory

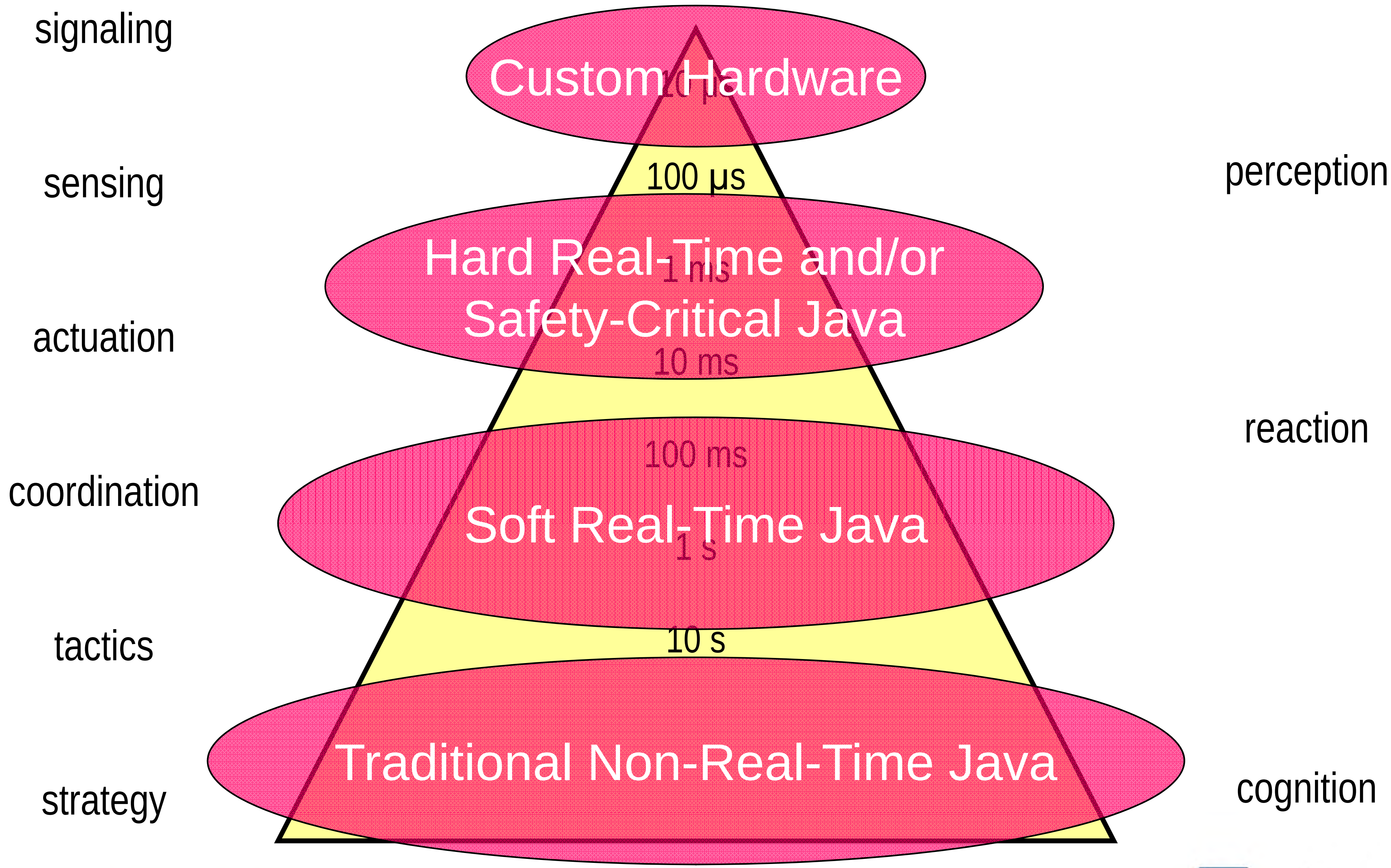
Efficient and robust integration of legacy and Java

- Most real-world systems are comprised of many different technologies, each targeting different needs and making different compromises
- Typical real-time systems address a combination of hard real-time and soft real-time constraints
- The option of using C for low-level hard real-time code and Java for higher-level soft real-time code is less desirable:

  Performance overhead due to marshaling of data

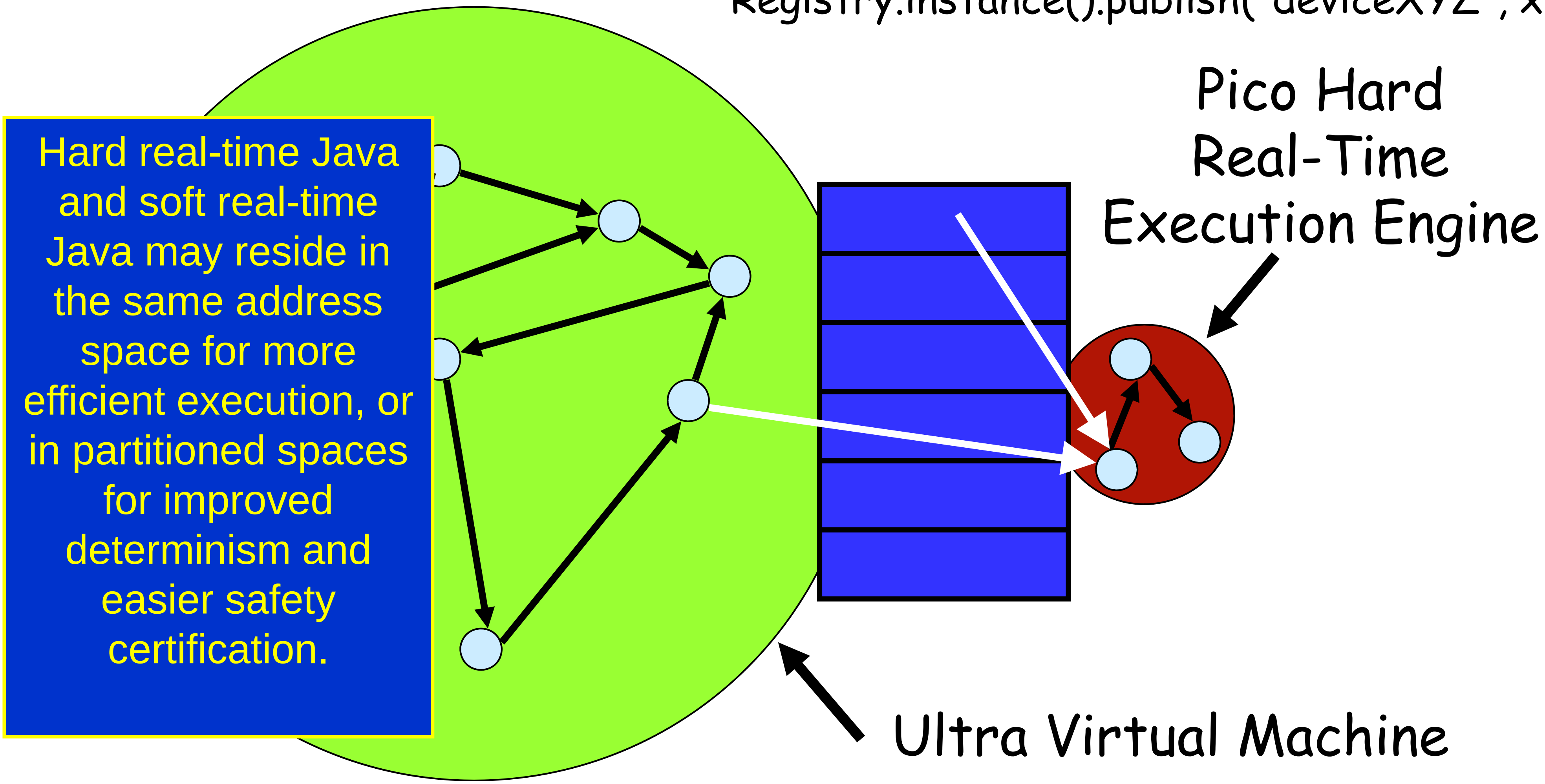
  Reliability and maintenance challenges because C code compromises integrity of Java security system

Real-Time Pyramid (Hierarchies and Layers)



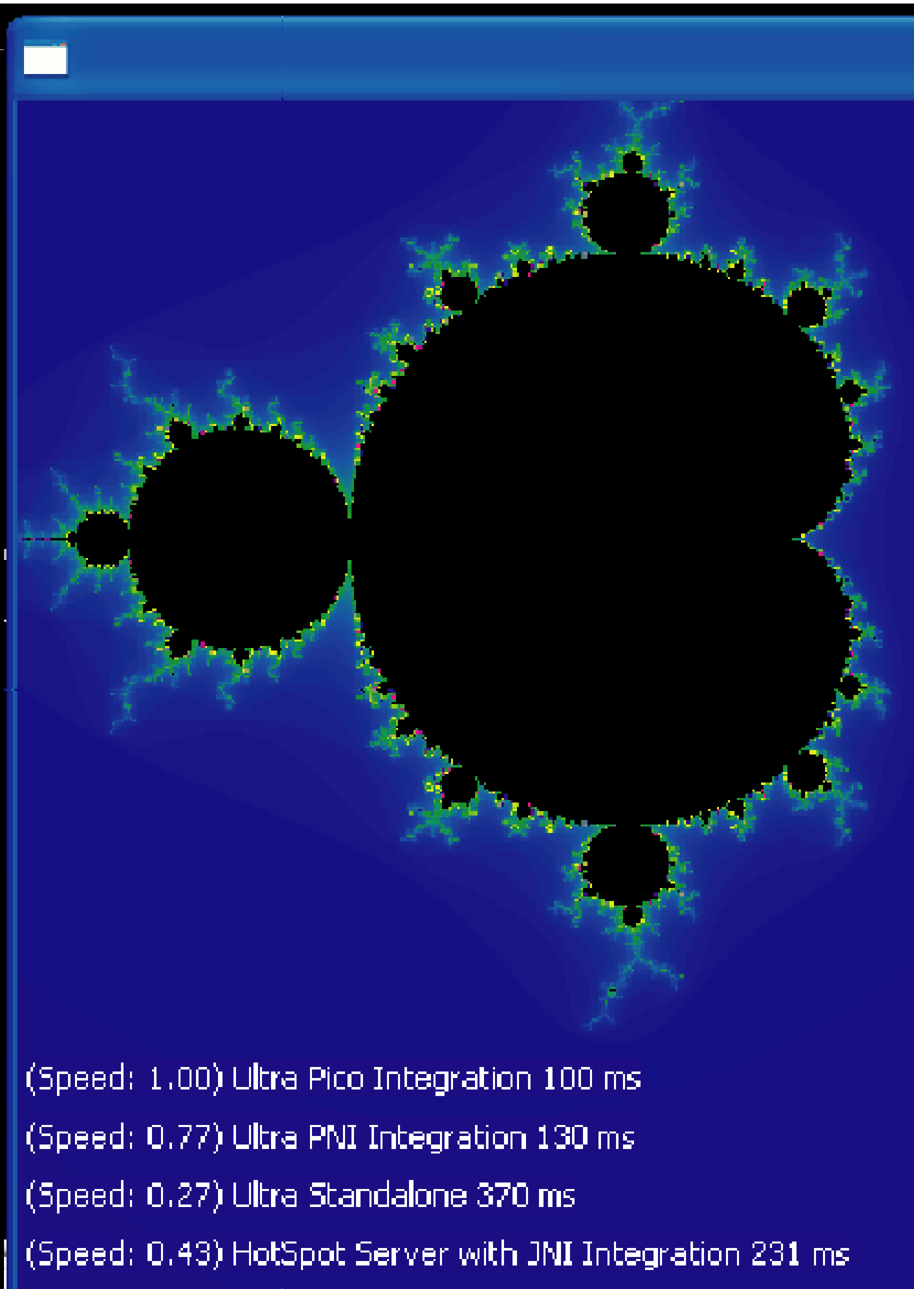
Cooperating HRT Components

Registry.instance().publish(“deviceXYZ”, x);



jvmadapter.Registry.lookup(“deviceXYZ”);

All Java Solution Twice as Fast as C/Java!



- An ESC Silicon Valley 2007 demonstration: all Java application runs over twice as fast as hybrid solution comprised of Java and C
- Further benefit: all Java solution is easier to develop and maintain, offering superior separation of concerns

Summary

- Real-time Java has been successfully deployed in various commercial and defense applications
- In successful projects, the choice to use Java has reduced certain risks and demonstrated concrete benefits
- However, the mere act of choosing to use Java does not guarantee successful deployment, risk elimination, or software engineering benefits
- Effective use of real-time Java requires effective management in order to address relevant success factors

Acronyms

- BAE: British Aerospace Engineering
- COTS: Commercial off-the-shelf
- DSL: Digital Subscriber Line
- ESC: Embedded Systems Conference
- FELIN: Fantassin à Equipements et Liaisons Intégrées (French)
- FIFO: First in, first out
- FTTP: Fiber to the Premises
- GPON: Gigabit Passive Optical Network
- J2SE: Java 2 Standard Edition
- JSR: Java Specification Request
- J-UCAS: Joint Unmanned Combat Air Systems
- MDS: Mission Data System
- NASA: National Aeronautics and Space Administration
- RTSJ: Real-Time Specification for Java
- SONET: Synchronous Optical Network
- UAV: Unmanned Aerial Vehicle
- UK MoD: United Kingdom Ministry of Defence